VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Desková hra TransAmerica

TransAmerica Board Game

2016 David Damek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Damek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Desková hra TransAmerica
TransAmerica Board Game

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit elektronickou verzi deskové hry TransAmerica. Výsledkem bude klientská aplikace určená pro hráče a serverová aplikace, která bude průběhy her řídit. Práce se bude zabývat také návrhem automatického protihráče. Součástí bude rovněž databáze s informacemi o uživatelích (hráčích), hrách, turnajích, výsledcích a jednotlivých tazích.

Práce bude obsahovat následující body:
1. Stručný popis pravidel hry.
2. Rešerše existujících řešení.
3. Návrh architektury řešení.
4. Návrh síťového protokolu pro komunikaci herní aplikace a serveru.
5. Návrh databáze.
6. Návrh a implementace herní aplikace.
7. Návrh a implementace herního serveru.
8. Zapracování prvků umělé inteligence a vytvoření automatického protivníka.
9. Srovnání s existujícími implementacemi.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

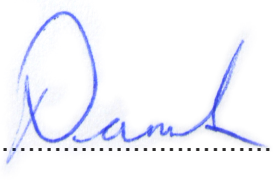
doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, z kterých jsem čerpal.

V Ostravě 29. duben 2016



.....

Tímto chci poděkovat Ing. Petru Lukášovi, jakožto vedoucímu práce, za rady a odbornou pomoc, kterou mi vždy neváhal poskytnou, vstřícný přístup a kvalitní odborné vedení.

Abstrakt

Práce je zaměřená na návrh a implementaci počítačové verze deskové hry TransAmerica. Cílem práce je naprogramovat velmi přesnou kopii hry s využitím grafových algoritmů v programovacím jazyce C#. Následně návrh síťového protokolu pro možnou komunikaci více hráčů přes server.

Klíčová slova: TransAmerica, grafové algoritmy, C#, server, klient, databáze

Abstract

This work is focued on design and implemetation of board game TransAmerica to interactive vision. Main goal is to programme very precise copy of the game with of use graph algorithms in C# programing language. Then design net protocol for potential net game communication for more players.

Keywords: TransAmerica, graph algorithms, C#, server, client, database

Seznam použitých zkratek a symbolů

TCP	–	Transmission Control Protocol
IP	–	Internet Protocol
OS	–	operační systém
GUI	–	Graphical User Intreface
MVC	–	Model-View-Controller
JSON	–	JavaScript Object Notation
atd.	–	a tak dále
UML	–	Unified Modeling Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Obsahem této práce je seznámit čtenáře s problematikou vývoje počítačové hry TransAmerica, která vychází z deskové hry stejného jména. Tato desková hra vznikla na začátku 21. století a vzápětí dostala několik ocenění, jakožto nejlepší hra pro začátečníky [1]. Pro velkou oblibu hry vzniklo i několik rozšíření, jako je například TransEurope. Později se zapojila komunita hráčů, která vytvořila řadu dalších neoficiálních rozšíření, která je možné nalézt na Internetu a stáhnout.

Proč vůbec tato počítačové hra vznikla? Hra TransAmerica je založena na grafové síti, která je složena ze základních stavebních elementů grafu pomocí vrcholů a hran. Z tohoto důvodu je hra vhodná pro zpracování v elektronické podobě. Bohužel v obchodech už je hra velmi obtížně sehnatelná a nejlepší řešení pro její zachování bylo vytvořit kvalitní počítačovou hru, která ji bude věrně napodobovat. Další výhodou a motivací pro její tvorbu byla možnost využití síťových prostředků pro vzdálenou hru více hráčů a v neposlední ředě možnost hrát proti umělému protivníkovi, který by dokázal napodobit hru člověka. Klasická desková hra tyto možnosti neumožňuje.

První část této práce obsahuje stručná pravidla pro lepší pochopení hry. Následuje kapitola 3, ve které jsou popsáná již existující řešení a jejich jednotlivá srovnání. V kapitole 4 je popsán rozbor principů souvisejících s logikou hry, tedy základní algoritmy pro práci s grafem a analýza herní mapy. Po této kapitole následuje kapitola 5, která se zabývá samotným návrhem a implementaci klientské části hry. Následně navazuje kapitola 6 a 7. V kapitole 6 se nachází návrh databáze a v kapitole 7 návrh a implementace serveru. Tato kapitola je rozšířena o popis protokolu komunikace, který zajišťuje propojení klienta a severu.

2 Pravidla hry TransAmerica

2.1 Mapa

Mapa se skládá z měst, bodů a hran, na které může hráč stavět železnici. Jednotlivé hrany jsou značené jednoduchou nebo dvojitou čárou. Ve hře to znamená, že jednoduchá hrana stojí jeden bod a dvojitá dva body. Hrana za dva body se vyskytuje na obtížně stavitelných místech, jako jsou hory a řeky. Stavba tunelů a mostů je totiž i v reálném světě velmi nákladná. Dále je na mapě vyznačeno 35 měst rozdělených do skupin po sedmi, kde každá skupina je reprezentována jedou z pěti základní barev – červenou, zelenou, modrou, žlutou nebo oranžovou [2].

2.2 Cíl hry

Na začátku hry si hráč vylosuje pět měst, které ostatní hráči nesmí vidět. Cílem hry je následně propojit pomocí železnice tato města. Jakmile se to nějakému hráči podaří, je herní kolo ukončeno a ostatním hráčům se udělují trestné body. Hráč obdrží trestný bod za každý nepostavený úsek železnice, který by vedl k propojení jeho pěti měst. Hrany se počítají podle těchto pravidel:

- 1 bod za každou nepostavenou železnici, která vedla přes rovinatý terén, tedy jednoduchá čára.
- 2 body za každou nepostavenou železnici, která vedla přes hory nebo řeky. Na mapě jsou značené dvojitou čárou.

Hra obvykle končí po několika kolech, když jeden z hráčů překročí závoru, která ukazuje maximální možný počet trestných bodů.

2.3 Průběh jednoho kola hry

Hra začíná tím, že každý hráč postupně položí startovní kámen na jeden z vrcholů grafu, na kterém není doposud postaven kámen jiného hráče. Kámen může hráč založit i ve městě. Správná volba startovního kamene je velmi důležitá, protože od tohoto bodu se začíná stavět železnice. Po postavení všech základních kamenů začíná hra. Hráč má každé kolo dostupné dva stavební body, pomocí kterých může postavit jeden nebo dva úseky železnice v závislosti na hodnotě hrany grafu.

Pravidla pro umísťování železnice jsou následující:

- Hráč musí stavět jen od svého základního kamene.
- Hráč nesmí stavět na cizí železniční, síť na kterou není sám napojen.
- Hráč se může napojit na cizí síť, a tím ji využívat jako vlastní.

Kolo se ukončí, jakmile první hráč propojí všech svých pět měst nebo se vyčerpá maximální limit použitelných hran (84). Následně se dopočtou trestné body pro všechny hráče. Tímto je jedno kolo hry ukončeno a pokračuje se dalším. Hrany a základní kameny se odstraní, města se znovu náhodně roztřídí mezi hráče a začíná se opět od položení základního kamene.

3 Existující řešení hry a srovnání

Táto část práce se zaměřuje na již existující implementace hry TransAmerica.

3.1 TransSib

První hra, kterou lze volně stáhnout nese název **TransSib** [4]. Tato hra je naprogramovaná v jazyce Java a její zpracování je relativně kvalitní. Výhodou je přiložený program pro interaktivní vytvoření mapy a libovolná přenositelnost na všechny zařízení s podporou platformy Java. Nevýhodou je, že nemá otevřený kód a musí se instalovat. Neumožňuje tedy hru dále rozšiřovat komunitou. TransSib také neumožňuje hru po síti. Grafická stránka (viz obrázek 1) hry je již zastaralá a vývoj této hry je již několik let ukončen.

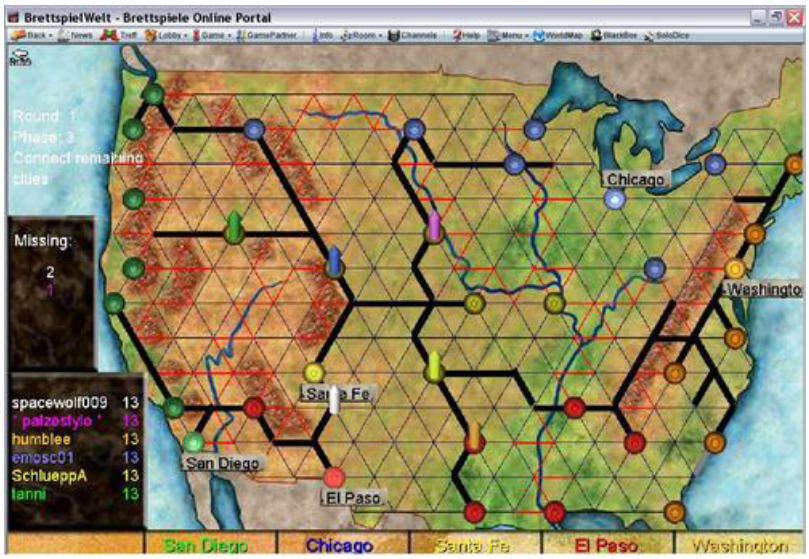
Obrázek 1: TransSib

Shrnutí kladů a záporů hry:

+ Přenositelnost na zařízení s podporou platformy Java.
+ Editor pro vytváření map.
+ Automatický protivník s více obtížnostmi.
– Zastaralá grafická stránka hry.
– Neumožňuje síťovou hru.
– Uzavřený kód.

3.2 Brettspielwel TransAmerica

Další elektronická implementace hry TransAmerica je dostupná ve webovém prohlížeči jako Java applet. Je zakomponovaná do německého herního serveru Brettspielwelt [3], který nabízí přes 50 deskových her. Nutnost je mít v prohlížeči nainstalovanou Javu a povolit bezpečnostní výjimku. Nutná je také registrace na herním serveru. Tuto hru se nám i po nejnovější aktualizaci Java pluginu nepodařilo spustit. Zásadní nevýhodou je tedy problematické spuštění. Hra má jen jednu mapu a neumožňuje hru v offline režimu. Hra nemá zabudovaného automatického protivníka. Obrázek ze hry 2:



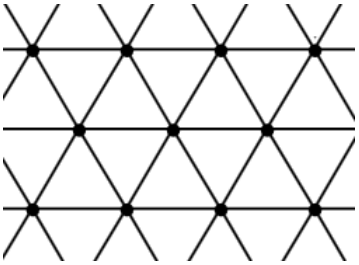
Obrázek 2: Brettspielwel TransAmerica

Shrnutí kladů a záporů hry:

+ Hra je dostupná z webového prohlížeče.
+ Herní server nabízí více druhů deskových her.
+ Přenositelnost na zařízení s podporou platformy Java.
– Absence automatického protivníka.
– Problematická funkčnost.
– Neumožňuje hru v offline režimu.

4 Rozbor principů souvisejících s logikou hry

Herní logika je postavena na práci s grafem. V následujících podkapitolách bude tato problematika postupně rozepsána.



Obrazek 3: Ukázka části grafu podkladové mapy

4.1 Grafové znázornění mapy

Neorientovaný, souvislý, rovinný graf mapy je složen z vrcholů a ohodnocených hran, které jsou koncipovány do trojúhelníkové sítě (viz obrázek 3). Tyto hrany nabývají hodnot 0, 1 a 2.

Bližší vysvětlení pojmů grafu:

- Neorientovaným grafem označujeme graf, jehož hrany jsou dvouprvkové množiny. Nemají daný směr. Tudíž dvojice (x, y) a (y, x), kde x a y jsou vrcholy, označují stejnou hranu.
- Souvislým grafem označujeme graf, ve kterém platí, že pro každé dva vrcholy x, y existuje sled z x do y.
- Rovinný graf je takový graf, pro který existuje rovinné nakreslení, v němž se žádné dvě hrany nekříží.

4.2 Algoritmy pracující s grafem

Při návrhu mapy bylo nutno nalézt vhodné algoritmy, které pracují s grafem. Jedním z prvních bylo automatické nalezení nejkratší cesty grafem. Přímo v samotné hře je nutné vypočítat, kolik hran chybělo hráči k propojení pěti měst. Pokud by hra neměla implementovaného umělého protivníka, byl by algoritmus použit jen v tomto případě. Hra ovšem tuto možnost nabízí a algoritmus je použit ještě pro další dva případy:

1. Výpočet nejlepší polohy pro položení základního kamene.
2. Následné nalezení nejkratší cesty ze základního do měst.

V obou případech je na výběr z více druhů algoritmů. Nejznámější algoritmy pro nalezení nejkratší cesty jsou:

1. Bellman-Fordův,
2. Floyd-Warshallův,
3. Dijkstrův.

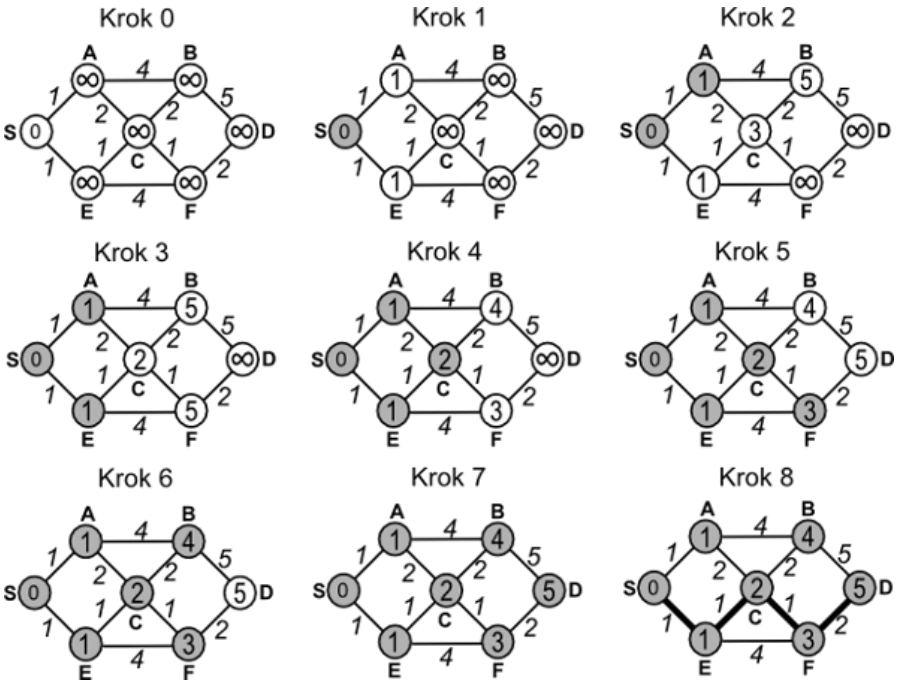
První dva zmíněné algoritmy se používají, kdyby graf obsahoval i záporně ohodnocené hrany a proto jsou i časově náročnější na výpočet.

4.2.1 Hledání nejkratší cesty

Jedním z nejefektivnějších algoritmů pro nalezení nejkratší cesty z vrcholu A do všech ostatních vrcholu grafů je Dijkstrův algoritmus. [5] Jeho složitost je dána vzorcem:

$$O(|V|^2 + |E|) \tag{1}$$

kde $|V|$ je počet vrcholů a $|E|$ počet hran. Jedinou nevýhodou tohoto algoritmu, že graf nesmí obsahovat jakoukoli hranu záporné délky. Graf ve hře má jen kladně nebo nulové hrany, tzn. má jen nezáporně ohodnocené hrany, a proto je tento algoritmus pro požadavky hry ideálním řešením.



Obrázek 4: Nalezení cesty

Tabulka 1: Tabulkový výpis

Vrcholy	S	A	B	C	D	E	F
Krok 0	0	∞	∞	∞	∞	∞	∞
Krok 1	0	1	∞	∞	∞	∞	∞
Krok 2	0	1	∞	∞	∞	1	∞
Krok 3	0	1	5	3	∞	1	∞
Krok 4	0	1	4	2	∞	1	5
Krok 5	0	1	4	2	∞	1	3
Krok 6	0	1	4	2	5	1	3
Krok 7	0	1	4	2	5	1	3

Hledání cesty grafem je znázorněno na obrázku 4 a tabulce 1. V kroku 1 se vybere výchozí vrchol a nastaví se na vzdálenost 0. K sousedním vrcholům se následně připočte hodnota hrany a hodnota aktuálního vrcholu. V kroku 2 se následně vybere nejmenší nenavštívený vrchol a opět ohodnotí sousedy. Takto algoritmus projde celým grafem až do kroku 7.

Výsledkem na obrázku 4 je nejkratší cesta z kteréhokoliv vrcholu A-F do základního bodu S. V kroku 7 máme plně ohodnocený graf. Cestu nalezneme zpětným průchodem například z bodu D do bodu S. Takováto cesta má délku 5 a je vyznačená v kroku 8 tučnou čárou.

Textový popis

Algoritmus je vysvětlen na obrázku 4 a následně pomocí pseudokódu 2.

Na vstupu funkce jsou tři základní proměnné. Na řádku 1 je množina všech vrcholů (značená *V* jako *vertices*), množina všech hran (značená *E* jako *edges*) a výchozí bod (značen *s* jako *source*).

Ke každému vrcholu se postupně přiřazuje číslo udávající jeho nejkratší cestu do vrcholu s. Před spuštěním hlavního cyklu se nastaví pro všechny vrcholy v z množiny V hodnota nekonečno (řádek 2) a uloží se do pole *dist* (řádek 3). Po průchodu všech vrcholů a nastavení základních hodnot se cyklus (řádky 8 - 18) ukončí. Pro výchozí vrchol *s* v poli *dist* se nastaví vzdálenost 0 (řádek 6). Pole vrcholů se uloží do pomocného pole N, ve kterém jsou uloženy všechny nenavštívené vrcholy (řádek 7).

V hlavním těle algoritmu se prochází pole *N* nenavštívených vrcholů, dokud toto pole není prázdné. Jako první se vybere nejmenší vrchol z pole *dist*. Následuje uložení tohoto vrcholu do proměnné *u* (řádek 9) a následně se odstraní z pole nenavštívených vrcholů Q (řádek 10). Vnořený cyklus projde všechny sousední vrcholy z množiny sousedů a opět vybere ten s nejmenší hranou propojující aktuální vrchol s vybraným. U vybraného vrcholu algoritmus spočítá délku hrany a hodnotu vrcholu *u* (řádek 12). Tímto průchodem se zjistí, jestli cesta z vybraného vrcholu je kratší než ta, která je ve vrcholu uložena. Do pole *prev* se uloží vybraný vrchol v. Takto algoritmus projde celý graf. Zpětným průchodem od libovolného vrcholu nalezneme nejkratší cestu do výchozího bodu S.

Algorithm 1 Dijkstrův algoritmus

```
1:  function DIJKSTRA(E, V, s)
2:      for all vertex v in V do                          ▷ Inicializace.
3:          dist[v] ← infinity                ▷ Neznámá vzdálenost z počátku s do vrcholu v.
4:          prev[v] ← undefined   ▷ Předchozí vrchol na nejkratší cestě z počátku s k cíli.
5:      end for
6:      dist[s] ← 0                                       ▷ Vzdálenost s rovná 0.
7:      N ← V                          ▷ Množina všech dosud nenavštívených vrcholů.

8:      while N is not empty do                  ▷ Hlavní smyčka průchody grafem.
9:          u ← vertex in N with min dist[u]     ▷ Najde vrchol s nejmenší vzdáleností.
10:         remove u from Q

11:         for all neigbor v of u do
12:             alt ← dist[u] + length(u, v)          ▷ V 1. smyčce cyklu u je s d[u] = 0.
13:             if alt ≥ vzlenost[v] then
14:                 dist[v] ← alt
15:                 prev[v] ← u
16:             end if
17:         end for
18:     end while
19: end function
```

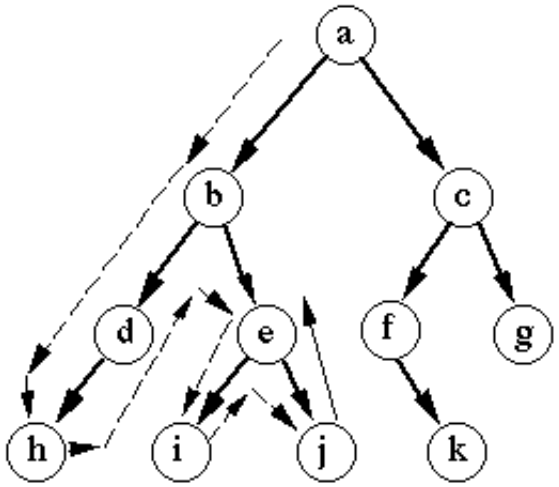
4.2.2 Průchod grafem do hloubky

Dalším problémem pro správné fungování hry bylo vyhledání všech dostupných hran, na kterých může hráč stavět železnici. Jak už bylo zmíněno v pravidlech v kapitole 2, hráč může stavět pouze na trať, která je propojena s jeho základním kamenem. Bylo tedy nutné vyhledat indukovaný podgraf, který tvoří souvislou komponentu. Podgraf musí obsahovat vrchol reprezentující základní kámen a hrany ohodnocené 0, tedy hrany, na kterých je postavená železnice. Všechny okolní hrany tohoto podgrafu se následně nastaví jako aktivní a může se na ně stavět.

Vhodným řešením pro tento problém se ukázalo použití algoritmu *prohledávání grafu do hloubky* [6] *DFS (depth-first search)*, který je jedním ze základních grafových algoritmů. Příklad průchodu je znázorněn na obrázku 5.

Princip algoritmu:

1. Nejprve označíme vrchol *s*,
2. poté označíme jednoho následovníka vrcholu *s*, pak jeho následovníky atd.
3. Až vyčerpáme všechny dostupné následovníky, vracíme se postupně zpět
4. a pokračujeme dalším následovníkem vrcholu *s*.



Obrazek 5: Průchod do hloubky

Pseudo kód implementace pomocí rekurze

Vstup: Graf G a vrchol v (v je základní kámen)

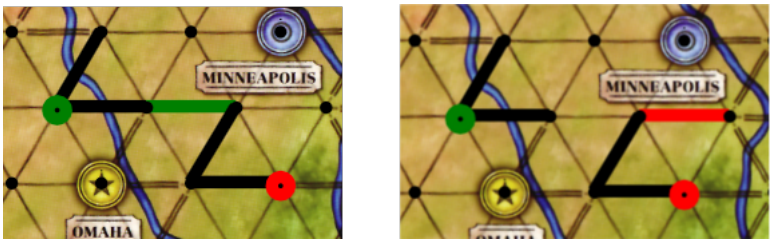
Algorithm 2 Průchod grafem do hloubky

```
1: function DFS(G, v)
2:     v ← discovered
3:     for all edges from v to w in G.adjacentEdges(v) do
4:         if vertex w is not labeled as discovered then
5:             recursively call DFS(G, w)
6:         end if
7:     end for
8: end function
```

Popis implementace ve hře

Algoritmus prohledávání do hloubky aplikujeme na herní mapu tak, že základní kámen hráče představuje vstupní vrchol. Algoritmus uvažuje pouze hranu s hodnotou 0, které představují postavenou železnici. Pokud je postavená, provede se další rekurzivní iterace pro všechny sousední vrcholy a zkontrolují se opět všechny hrany. Pokud hrana má velikost 1 nebo 2, nastaví se jako aktuální a hráč na ní může stavět. Od tohoto okamžiku se jedno vnoření rekurze ukončí.

Ukázka grafického znázornění dostupných hran ve hře je znázorněno na obrázku 6. Zelená hrana značí dostupnou kolej a červená nedostupnou. Výpočet se provedl ze zeleného startovního kamene. Červená je sice napojená na aktivní hranu, ale ta není napojená na trať zeleného hráče.



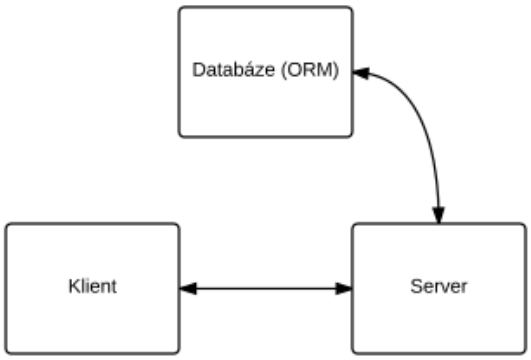
Obrazek 6: Ukázka postavení koleje

5 Návrh a implementace klienta

V této kapitole se nachází obecný návrh programu. K vizualizaci jsou použity diagramy UML [11].

5.1 Obecný návrh

Ze zadání práce vyplývá, že aplikace bude zajišťovat hru po síti a bude využívat databázi. Hru lze tedy v základu rozdělit na tři části: klient, server a databáze (viz obrázek 7).



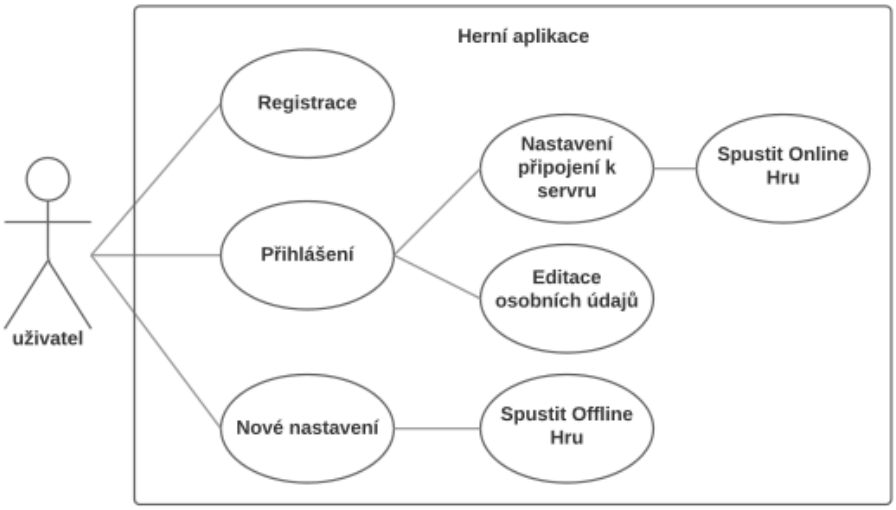
Obrazek 7: Základní rozložení projektu

- *klient* – zde se odehrává základní logika hry a interakce s uživatelem,
- *server* – zajišťuje komunikaci mezi jednotlivými klienty a spojení s databází,
- *databáze* – ukládá výsledky her, hráče a mapy.

Tyto tři části navzájem komunikují oběma směry. Například pokud bude chtít klient získat data z databáze musí nejprve zaslat požadavek na server a ten následně na databázi. Z databáze se vyberou patřičná data a zašlou se na server, který je opět přepošle na klienta. Pro implementaci aplikace bylo zvoleno vývojové prostředí .NET, konkrétně WPF.

5.2 Klient

Klient je základní a velmi důležitá část aplikace. Obsahuje kompletní herní logiku pro lokální hru a část logiky pro síťovou hru. Také se zde nachází všechny potřebné metody pro vytvoření mapy a grafické uživatelské rozhraní. Server tedy pouze synchronizuje data mezi klienty. Další nepostradatelnou součástí je přístup a komunikace se serverem při síťové hře.



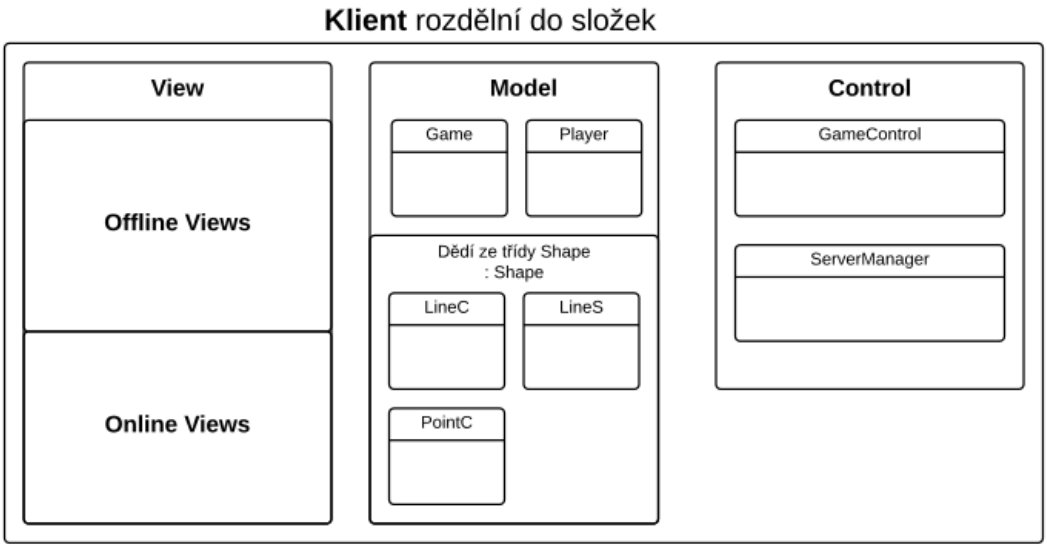
Obrázek 8: Případy užití – klient

Klientská část aplikace je řešena pro dvojí přístup. To ovšem záleží na typu hry. Pokud se jedná o lokální hru bez využití serveru, je přístup otevřen všem uživatelům bez nutnosti registrace nebo připojení k Internetu. V tomto režimu se neukládají žádná data do databáze. V druhém režimu pro síťovou hru je uživatel nucen se přihlásit pod svým unikátním jménem. Je nutné stabilní připojení k síti a tento režim hry je přístupný pouze registrovaným uživatelům. Jednotlivé případy užití jsou znázorněny na obrázku 8.

Protože aplikace nabízí více režimů her, je klient rozložen na síťovou a lokální hru a ty využívají společné třídy. Celá architektura je navržená pomocí návrhového vzoru Model-View-Controller [12]. To zjednodušeně znamená, že v části *model* jsou uložena data a ve view se volají jednotlivé události na reakce hráče. Tyto dvě části nejsou navzájem propojené, ale využívají takzvaný *controller*, který je prováže. Výhoda je ta, že logika hry je oddělaná od dat. Celkové rozložení klienta je zobrazeno na obrázku 9.

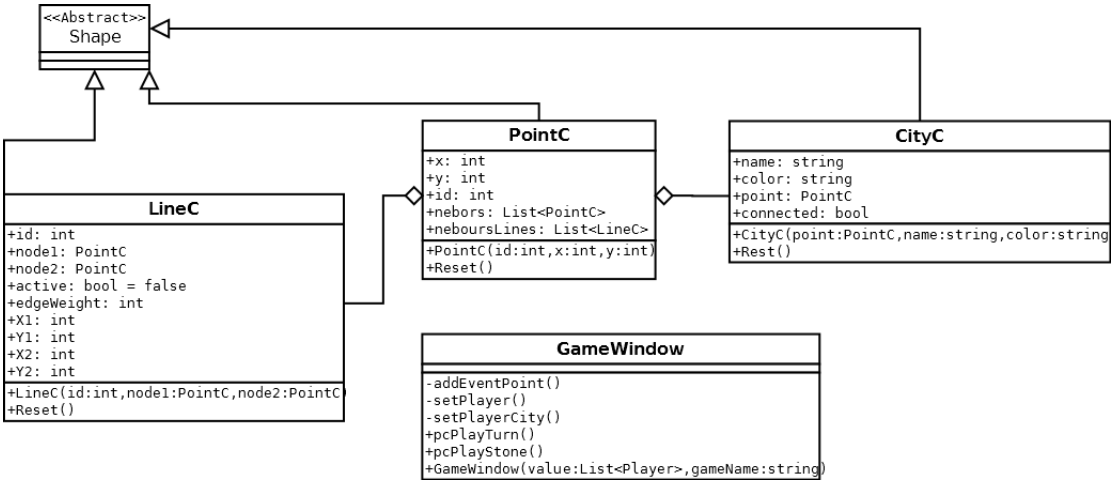
- *Model* – obsahuje reprezentaci grafu a instance hráčů,
- *View* – poskytuje grafická okna,
- *Control* – poskytuje herní logiku.

Jednotlivé části jsou blíže rozepsané v následujících podkapitolách.



Obrázek 9: Rozložení klienta

5.2.1 Model – mapa a model grafu



Obrázek 10: Objektová reprezentace mapy/grafu

Hlavním prvkem herního okna je mapa vložená do komponenty *Canvas*. Tato mapa se skládá z podkladového bitmapového obrázku, který se načítá z databáze přes server při síťové hře. Takto získaný obrázek má podobu bajtového pole a pomocí funkce *ToImage* se vytvoří bitmapový obrázek. K tomuto účelu se využívá *IOMemoryStream* a následně se nastaví na pozadí canvasu.

Další součástí mapy jsou hrany, vrcholy a města. Tyto prvky jsou reprezentovány geometrickým zápisem. Protože základní objektové tvary WPF mají jen omezené vlastnosti, bylo nutné použít dědičnosti ze třídy *Shape*. Nové tvary, tj. třídy, jsou rozšířené o atributy a metody, které jsou nezbytné pro reprezentaci grafu (viz obrázek 10).

Vrcholy jsou vytvořeny pomocí třídy *PointC* a metody *DrawGeometry*. Nejdůležitější atributy, které každý objekt vlastní jsou tyto:

Vrchol – PointC

- *id* – jedinečný identifikátor,
- *x* – souřadnice x na elementu *Canvas*,
- *y* – souřadnice y na elementu *Canvas*,
- *neighbours* – seznam sousedních bodů.

Hrana – LineC

- *active* – určuje, zda je hrana postavená
- *available* – tato hodnota určuje, jestli může na hráč tuto hranu stavět kolej,
- *node1* – první vrchol hrany, reference na určitý objekt ze třídy *PointC*,
- *node2* – druhý vrchol hrany, reference na určitý objekt ze třídy *PointC*.

Město – CityC

- *connected* – určuje, jestli je město propojeno,
- *color* – barva slouží pro správné přidělení měst,
- *point* – reference na bod z *PointC*, na kterém je město umístěno.

Metoda *Reset* nastaví objektu výchozí hodnoty. Tato metoda se volá při načtení nového kola, kde je nutné vykreslit výchozí graf. Mapa se následně vykreslí při vytvoření okna *GameWindow* pomocí metody *DrawMap*. V cyklu se projdou všechny objekty ze třídy *Game* a vykreslí se do elementu Canvas.

Třída *Game* je navržena podle návrhového vzoru *jedináček* (*singleton*) [8]. Existuje pouze jediná instance této třídy a jsou v ní uložena všechna data o mapě. Při offline hře se mapa načte z databáze a při offline hře se data načítají z textového souboru. Jednotlivé množiny objektů jsou uloženy v datové struktuře *List*. Celkem jsou čtyři a jejich názvy jsou tyto:

- *listPoints* – množina všech objektů třídy PointC,
- *listLines* – množina všech objektů třídy PointC,
- *listCity* – množina všech objektů třídy CityC,
- *players* – množina hráčů ze třídy Player.

5.2.2 View – grafické uživatelské rozhraní

Celkové zpracování a grafická podoba oken byla vytvořena pomocí návrháře WPF, který umožňuje interaktivní vkládaní ovládacích prvků, přičemž na pozadí se vytváří reprezentace v podobě XAML syntaxe.

Pro offline hru jsou určená okna:

- *GameWindow* – herní okno,
- *NetMainGameWindow* – okno pro nastavení hry.

Pro offline hru jsou:

- *LoginWindow* – okno pro přihlášení,
- *AccoutStatistics* – okno se záznamy jednotlivých her,
- *NetMainGameWindow* – herní okno.

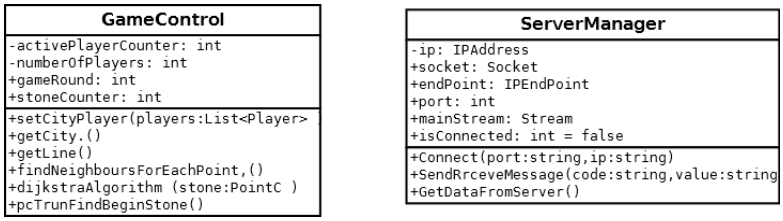
Ukázku jednoho z oken vidíme na obrázku 11. Další ukázky map a GUI je možné nalézt v příloze A.

Obrázek 11: TransAmerica

V pravém okraji jsou základní informace – který hráč je na tahu a kolik mu zbývá bodů na stavbu. Dále je zde uveden celkový počet kolejí, které jsou ještě k dispozici a herní kolo. Pod těmito informacemi se nachází panel se seznamem 5-ti měst hráče. Konec tahu se provádí tlačítkem *konec tahu*. Ve spodní liště se nachází seznam všech přihlášených hráčů ve hře a panel pro posílaní zpráv ostatním hráčům.

5.2.3 Controller

Posledním článkem návrhového vzoru MVC je takzvaný *Controller*.



Obrázek 12: Diagram tříd Controller

V aplikaci zajišťuje řízení hry, výpočty nad grafem, automatického protivníka, ale i samotnou komunikaci se serverem. [1] To vše zajišťují pouze dvě třídy (viz obrázek 12):

- *GameControl* – obsahuje herní logiku,
- *ServerManager* – zajišťuje komunikaci se serverem.

Třída *GameControl*, jak už bylo zmíněno, obsahuje hlavní logiku hry. Po zapnutí hry se spustí základní metody. Metoda *setCityPlayer* náhodně rozdělí města hráčům podle barev. Třída *GameControl* následně obsahuje metody pro získání dat ze třídy *Game*, například *getLine* nebo *getCity*. Pro graf se spustí důležitá metoda *findNeighboursForEachPoint*, která, jak už z názvu vyplývá, přiřadí každému bodu jeho sousední body. Tato metoda je velmi důležitá pro algoritmy, které prochází grafem. Algoritmické řešení průchodu grafem do hloubky je v metodě *path* a Dijkstrův algoritmus v metodě *Dijkstra algorithm*.

5.3 Automatický protivník

Naprogramování umělého protivníka bylo jedním z obtížnějších úkolů této práce. Bylo potřeba ošetřit hodně případů špatného zahrání tahu, výpočtu cesty nebo nalezení vhodného základního kamene. Automatického protivníka je možné nastavit pouze v offline hře, přičemž takovýchto protivníků může být v jedné hře až pět. Každá hra musí mít ovšem alespoň jednoho fyzického hráče. Logiku pro výběr základních kamenů a výběr tahů zajišťuje Dijkstrův algoritmus, který je blíže popsán v kapitole 4.2.1 Počítač začne svůj tah, pokud se zjistí, že je na tahu hráč s parametrem PC. Pokud ještě není postaven základní kámen, spustí se metoda *pcPlayStone* ze třídy *GameControl*. Zdrojový kód metody můžeme vidět ve výpisu 1.

[1] Při síťové hře je část logiky přesunuta na server pro zaručení konzistence data na všech klientech.

```
1  public PointC pcTrunFindBeginStone() {
2     int minPoint = 0;
3      if (!disableStone) {
4        foreach (CityC city in getActivePlayer().CityConect) {
5           dijksterAlroritmPCStone(city.point);
6         }
7         cityCounterForStone = 0;
8         int minPointDist = 1000;
9
10        foreach(int p in newListPoints){ {
11         if (p < minPointDist) {
12               minPointDist = p;
13                minPoint = newListPoints.IndexOf(p);
14            }
15        }
16        newListPoints.Clear();
17        return getPoint()[minPoint];
18     }
19    return null;
20 }
```

Výpis 1: Ukázka nalezení základního kamene

Textový popis metody

Metoda hledá vrchol, který má do všech pěti měst nejmenší vzdálenost.

V tomto případě by bylo možno použít klasický Dijkstrův algoritmus, který by se postupně provedl na každém vrcholu grafu a daný vrchol by si zaznamenal součet délek k městům. Po takovémto průchodu všemi vrcholy by se získal vrchol s nejmenším součtem cest do daných pěti měst, které musí počítač propojit. Časová složitost je v tomto případě úměrná počtu všech vrcholů grafu plus výpočet Dijkstra algoritmu na každém z nich.

Vhodnějším řešením bylo udělat opačný postup. Projít graf pouze z pěti vrcholů, a to z měst, která musí počítač propojit. Je ovšem nezbytné při každém průchodu grafem si délky cest k vrcholům poznamenat. Pokud by se tomu tak nestalo, vybral by se pouze nejmenší bod z posledního průchodu. Touto malou změnou se velmi omezila náročnost celého výpočtu a zmenšila se časová prodleva na minimum.

Hodnota *minPoint* na řádku 2 určuje index vybraného vrcholu. Následně proběhne kontrola, jestli není kámen již postaven. Cyklus na řádku 4 projde všech pět měst aktivního hráče. Další cyklus na řádku 10 pouze projde všechny body ohodnoceného grafu a vrátí vrchol s nejmenší hodnotou (řádek 17). Po ukončení metody se automaticky ukončí první tah počítače.

V dalším kole jsou základní kameny postaveny a v tomto případě se opět využije Dijkstrova algoritmu, ale tentokrát na výpočet pro nalezení nejkratší cesty. Metoda, která

vrátí množinu hran se nazývá *pcTurn*. Od základního kamene se spustí metoda *dijkstraAlgorithm*. Vstupem této metody je kámen aktuálního hráče a výstupem je ohodnocený graf (viz výpis 2).

```
1 foreach(CityC city in getActivePlayer().CityConect){
2   if (!city.conected && city.point.dist < pomVal) {
3     pomVal = city.point.dist;
4     pointConnect = city.point;
5   }
6 }
7 List<PointC> pathPointMy = pathCall(pointConnect);
```

Výpis 2: Ukázka výběru města k propojení

Na Výpisu 2 vidíme část metody *pcTrun*, která vyhodnotí město s nejmenší vzdáleností k základnímu kameni, které ještě není propojeno. Následně se nad tímto bodem provede metoda *pathCall*, která hledá sousedy s nejmenší vzdáleností, dokud nenarazí na vzdálenost nula. To značí, že je bod již napojen na základní kámen. Problém nastal při průchodu postavenou tratí, která má v ohodnoceném grafu stejnou hodnotu, protože postavená hrana má velikosti nula. Algoritmus se v tomto případě zacyklil, protože našel sousední vrchol se stejnou vzdálenosti a nevěděl kterým směrem se vydat. Tento problém vyřešilo zapamatování si již navštíveného vrcholu. Tímto ošetřením nemá algoritmus přístup na předchozí vrcholy.

Podle počtu stavebních bodů, které má počítač k dispozici, postaví úsek železnice z množiny hran, kterou vypočítaly předchozí kroky.

5.4 Komunikace se serverem

Kompletní komunikaci zajišťuje třída *ServerManager* v části Contoller. Data se posílají pomocí objektu typu *NetworkStream*. Po úspěšném připojení můžou nastat dva případy odeslaní zprávy na server.

Prvním z nich je časová smyčka, která si vyžádá aktuální data hry každou sekundu. Podle kódu server určí, kterou funkci má vykonat. V tomto případě se vrátí informace o všech hráčích, hranách, všech zpráv z chatu a počtu postavených kolejí. Data jsou uložena v textovém formátu JSON [10], a proto se provede jejich deserializace na objekty. K serializace objektů na JSON a deserializaci JSON na objekty se využívá frameworku Json.NET [9].

Ukázka obsahu zprávy *getData*:

- kód zprávy – `GetData#`
- chat ve formátu JSON – `["All: Vitejte"]#,`
- hráči ve formátu JSON – `[{"CityConect":[],"stone":null,"stoneServer":"500", "onMove":"Ne","CityConectServer":[],"WinGame":"Ne","Color":"Červená", "ColorEN":"Red","Name":"admin","Comp":"Hrac","railNumber":2,"Score":"0", "pointsToWin":0,"Status":"Ne"}]#,`

- Všechny hrany ve JSON – `[{"active":false,"id":0,"value":1},{"active":false,"id":1,"value":1 ... {"active":false,"id":N,"value":1}]#,`
- Počet dostupných kolejí – `85`.

Druhý případ přijímání nebo odesílání dat je vlastní volání metod v průběhu různých fázi hry. Při odesílání nebo přijímání dat se struktura metody v zásadě neliší. V obou případech se čeká nějaká odpověď ze serveru, ať už v podobě dat nebo jen potvrzení o přijetí a zpracování požadavku. Jeden z příkladů takovéto metody vypadá následovně. Jedná se o položení základního kamene.

```
1  public bool StoneClickMessage(int id, string pName) {
2     string val = id + "#" + pName;
3     string code = "StoneSelect";
4     var massage = SendReceiveMessage(code, val);
5     if (massage != null) {
6        if (massage[0] == code) {
7           return true;
8        }
9        return false;
10    } else
11    return false;
12 }
```

Výpis 3: Ukázka výběru města k propojení

Vstupem této metody je identifikátor bodu, na který se postavil kámen a jméno hráče. Tyto dva parametry se spojí do jednoho textového řetězce odděleného speciálním znakem #, který značí oddělovač. Server pomocí tohoto znaku snadno data opět rozdělí. Kód *StoneSelect* je již zmíněný identifikátor pro server. Samotné zaslání a přijetí zprávy zajišťuje metoda *SendReceiveMessage*, která data zapíše do *NetworkStreamu* a následně přečte odpověď. Výsledkem této metody je zpráva serveru v podobě `StoneSelect` a `OK`. Pokud se tak nestane, nastal někde problém a to ošetří podmínky.

6 Databáze

Pro perzistentní uchovávání dat, která vznikají v průběhu online hry, je nejlepším řešením použít relační databázi. Databáze slouží pro uložení jednotlivých hráčů a jejich statistik. Kromě toho jsou v databázi uloženy i samotné herní mapy, kde každá mapa je reprezentována podkladovým obrázkem a specifikací grafu. Databáze byla navržena pomocí programu Oracle Datamodeler a implementována na Microsoft SQL Server 2014.

6.1 Tabulky databáze

Všechny entity a vztahy mezi nimi včetně příslušných atributů a kardinalit jsou znázorněny na obrázku 13. Databázové tabulky jsou popsány následovně:

Maps 2 – v této tabulce jsou uloženy všechny základní informace o mapě.

Tabulka 2: Maps

Název	Typ	Délka	Klíč	Null	Popis
map_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
name	varchar	25		ne	jméno mapy
maxRails	integer			ne	maximální počet kolejí v jednom kole hry
forMaxPlayer	integer			ne	maximální možný počet hráčů
image_fk	integer			ne	podkladový obrázek

Images 3 – podkladový obrázek pro mapu.

Tabulka 3: Images

Název	Typ	Délka	Klíč	Null	Popis
image_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
imageFile	BLOB			ne	obrázek v bajtové podobě

Points 4 – jednotlivé vrcholy grafu.

Tabulka 4: Points

Název	Typ	Délka	Klíč	Null	Popis
point_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
positionX	integer			ne	pozice bodu na mapě v ose X
positionY	integer			ne	pozice bodu na mapě v ose Y
Maps_fk	integer		FK	ne	cizí klíč tabulky *Maps*

Lines 5 – jednotlivé hrany grafu.

Tabulka 5: Lines

Název	Typ	Délka	Klíč	Null	Popis
line_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
lineCost	integer			ne	cena hrany s hodnotou 1 nebo 2
points_fk1	integer		FK	ne	cizí klíč tabulky *Points*, první vrchol hrany
points_fk2	integer		FK	ne	cizí klíč tabulky *Points*, druhý vrchol hrany

Cities 6 – města na herní mapě, která hráč musí propojit.

Tabulka 6: Cities

Název	Typ	Délka	Klíč	Null	Popis
city_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
color	varchar	10		ne	barva města
name	varchar	25		ne	jméno města
points_fk	integer		FK	ne	cizí klíč tabulky *Points*, pozice města

Players 7 — tato tabulka slouží pro uchování všech registrovaných hráčů a jejích osobních údajů.

Tabulka 7: Players

Název	Typ	Délka	Klíč	Null	Popis
player_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
name	varchar	25		ne	jméno hráče
password	varchar	25		ne	heslo
email	varcahr	50		ano	email pro případnou komunikaci s uživatelem

Players_Games 8 – vazební tabulka.

Tabulka 8: Players_Games

Název	Typ	Délka	Klíč	Null	Popis
game_pfk	integer		PFK	ne	pidentifikátor hry
player_pfk	integer		PFK	ne	identifikátor hráče ve hře

Games 9 – v této tabulce jsou všechny odehrané hry.

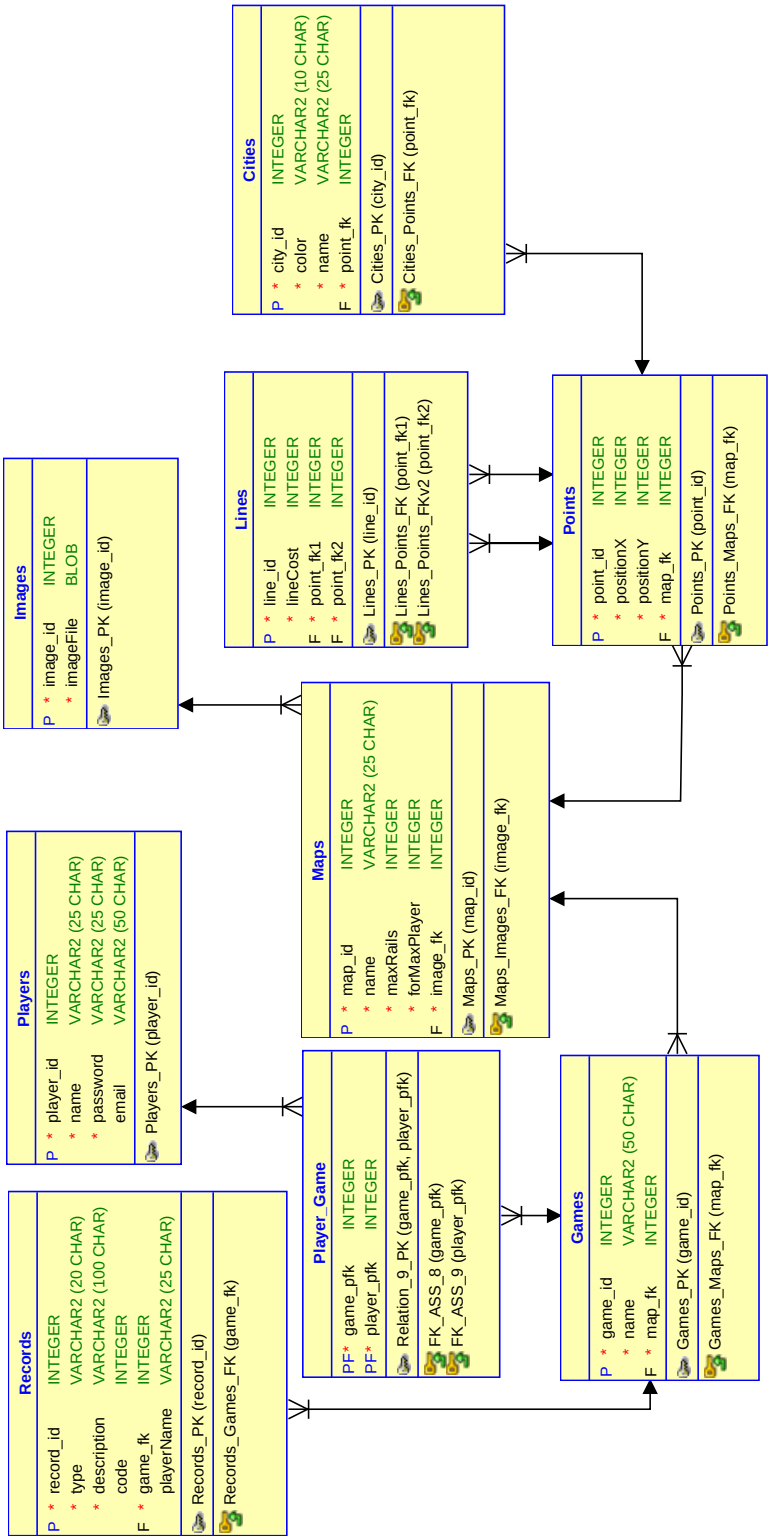
Tabulka 9: Games

Název	Typ	Délka	Klíč	Null	Popis
game_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
name	varchar	50		ne	název hry
map_fk	integer		FK	ne	odkaz na mapu, která byla použita pro hru

Records 10 – v této tabulce jsou uloženy všechny záznamy ze hry. Eviduje se chat, odehraná kola, hráči, položené kameny, položené hrany a výsledky.

Tabulka 10: Records

Název	Typ	Délka	Klíč	Null	Popis
record_id	integer		PK	ne	primární klíč, automaticky inkrementovaný
type	varchar	20		ne	typ záznamu pro rozlišení
description	varchar	100		ne	popis záznamu, například postavená kolej
code	integer			ne	určuje dodatečné rozlišení záznamu
game_fk	integer			ne	odkaz na hru
playerName	varchar	25		ano	jméno hráče, který provedl akci



Obrázek 13: Entitně relační diagram

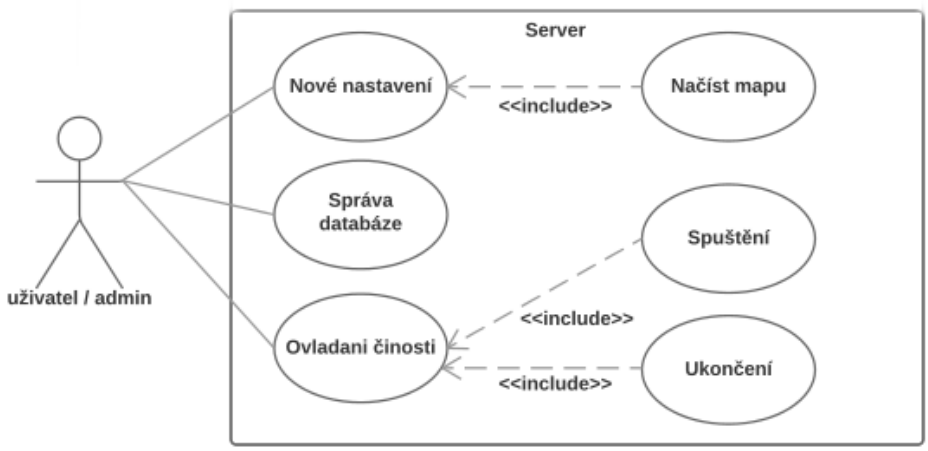
7 Návrh a implementace serveru

V této části se čtenář dozví o návrhu serveru pro hru TransAmerica. Hlavní motivací pro vytvoření serveru ve hře byla možnost hry s ostatními hráči, kteří jsou od sebe vzdálení. Také tato část aplikace disponuje grafickým uživatelským rozhraním s možností nastavení samotné hry. Nastavením hry se rozumí výběr mapy, počtu hráčů a portu TCP/IP, na kterém bude server naslouchat. Komunikaci mezi klienty, tedy hráči a serverem, zajišťuje protokol TCP/IP.

7.1 Návrh

7.1.1 Případy užití

Server je koncipován pro otevřený přístup. Z tohoto důvodu může kdokoliv vytvořit vlastní instanci serveru. Server řídí celou síťovou komunikaci a částečně i hru. Ze serveru je také možné přistoupit k databázi s možností spravování jejich dat s administrátorskými oprávněními. Jednotlivé případy užití jsou znázorněné v diagramu na obrázku 14.



Obrazek 14: Diagram případu užití – server

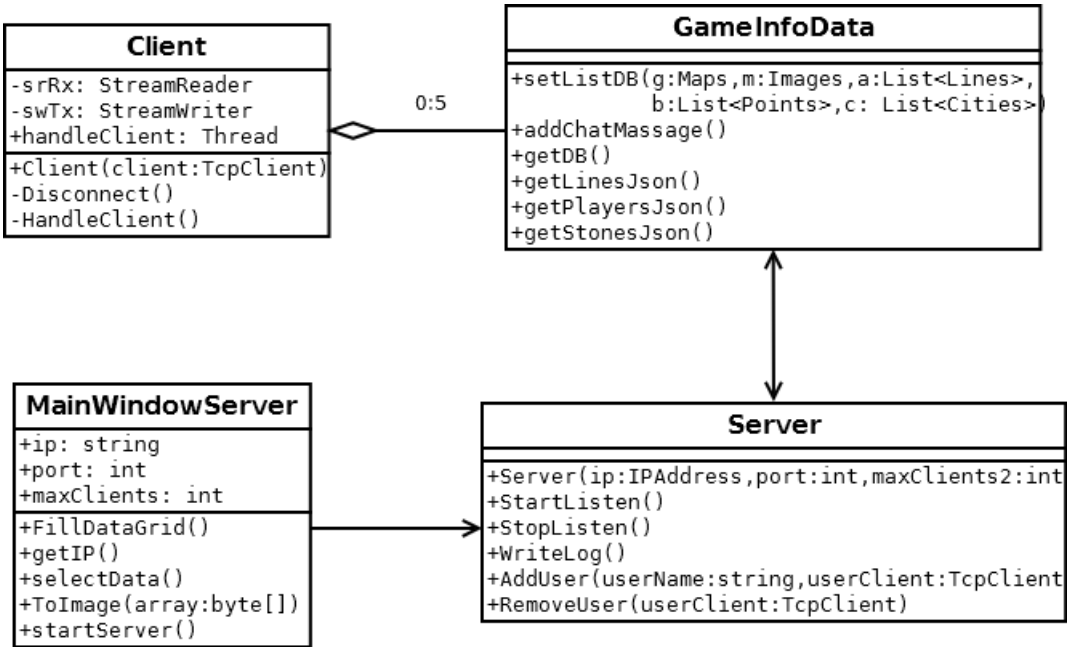
7.1.2 Stavová analýza

Možné stavy serveru jsou znázorněné stavovým diagramem, viz obrázek 15. V prvním kroku se server spustí a nastaví se požadované parametry pro vytvoření a řízení hry. Po nastavení se server spustí a přesune do aktivního (běžícího) stavu. Pokud není ukončen, pak zůstává v běhu do té doby, dokud je spuštěná hra.

Obrazek 15: Stavový diagram – server

7.1.3 Diagram tříd

V diagramu tříd jsou zobrazeny základní třídy aplikace serveru. Cílem je především rozložení funkcionality a vztahů mezi třídami. Diagram je doplněn popisem významu jednotlivých tříd.



Obrazek 16: Diagram tříd – server

- *MainWindowServer* je třída pro kód hlavního formuláře grafického prostředí serveru. Je zde nezbytná logika pro nastavení hry a serveru.
  - *FillDataGrid* výpis existujících map z databáze,
  - *getIP* přiřadí patřičnou IP adresu podsítě,
  - *selectData* vytáhne kompletní grafickou reprezentaci mapy z databáze podle id vybrané mapy,
  - *ToImage* převede bajtové pole na obrázek,
  - *startServer* zapne server a spustí novou instanci třídy *Server*.
- *Server* zajišťuje logiku fungování serveru. Zajišťuje propojení třídy *Client* a *GameInfoData*
  - *startListen* vytvoří nového listenera a začne naslouchat na přidělené IP adrese,
  - *StopListen* ukončí naslouchání,
  - *AddUser* přidá nového hráče.
- *GameInfoData* má pouze jednu instanci, kterou si drží po celou dobu spuštění serveru. V této třídě jsou uložena data o průběhu hry. V této třídě je i logika, která byla přesunuta z klienta a metody pro práci s datovým formátem JSON.
- *Client* je třída, která obstarává přihlášené hráče k serveru pomocí socketu. Vždy jedno vlákno klienta má jednoho hráče. Zprávy se posílají pomocí textového řetězce.

Server přistupuje k datum z databáze pomocí balíku tříd objektového relačního mapování, kde jedna každá třída reprezentuje jednu tabulku databáze.

7.2 Implementace

Po spuštění instance serveru správce vybere počet hráčů, kterým bude server naslouchat a následně spustí server. Potom se spustí metoda *Listen* ve třídě *Server*. Tato metoda pomocí cyklu while s podmínkou na začátku naslouchá, dokud se nepřipojí klient pomocí TCP protokolu. Tímto se automaticky vytvoří nový serverový klient, který běží ve svém vlákně. Od tohoto okamžiku začne metoda *HandleClient* ve třídě *Client* naslouchat na portu a očekává zprávy od klienta. Po přihlášení uživatele se spustí smyčka, která běží tak dlouho, dokud se neukončí.

7.3 Protokol komunikace

Návrh formátu zprávy pro komunikaci serveru a klienta je stěžejní pro fungování celé síťové komunikace. Server se nedotazuje na klienta, ale jen zpracovává požadavky od něj a zapisuje je do streamu. Formátem zprávy jsem se částečně inspiroval webovým HTTP protokolem. Formát zprávy vypadá následovně **{kód#hodnota1#....#hodnotaN}**, kde

- **kód** určuje typ zprávy, například **addUser** nebo **setLine**. Server si patřičnou zprávu pomocí kódu vyhodnotí a zpracuje. Seznam všech možných typů zpráv nalezneme v kapitole 7.4
- **hodnota 1 .. N** – v těchto hodnotách se posílají samotná data. Většinou je hodnota pouze jedna. Například pokud hráč položí kolej. V tomto případě se pošle v parametru identifikátor dané koleje.

Protože se zpráva posílá v textovém formátu, musí server vědět, kde je umístěn kód a jednotlivé hodnoty. Z tohoto důvodu jsem zvolil pro oddělení hodnot oddělovač #. Nutné je následně zabránit všem textovým vstupům jako je komponenta *textBox* zadávat hodnotu #. Mohlo by tak dojít k útokům ve stylu **Cross-site scripting (XSS)** [13].

Odpověď ze serveru je ve formátu **{kód#hodnota1#....#hodnotaN}**. Tato odpověď slouží jako informace pro klienta, že zpráva byla v pořádku přijata a zpracována. Ve stejném formátu se posílají i data, pokud si je klient vyžádá.

- **kód** určuje typ zprávy, kterou odeslal klient.
- **hodnota** definuje úspěšné doručení při posílaní dat na server. V tomto případě se za *hodnotu1* vloží **OK**. Při dotazu na získání dat se za hodnoty 1 až N uloží textové řetězce.

Všechny zprávy se ukládají do formátu JSON (JavaScript Object Notation) [9]. JSON je způsob zápisu dat (datový formát), nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována v objektech. Vstupem je libovolná datová struktura (číslo, řetězec, boolean, objekt nebo z nich složené pole), výstupem je vždy řetězec. Oproti XML (eXtensible Markup Language) je výsledný text poloviční velikosti a má objektovou reprezentaci. Formát textu je klíč:hodnota.

7.4 Kódy, které komunikují se serverem

- *PlayerLoginTransAmerica* – slouží pro přihlášení hráče na server. Posílá se jméno a heslo hráče.
- *PlayerRegister* – registrace hráče. Provede se zápis do databáze. Posílá se jméno a heslo hráče.
- *ChatSend* – přidání nového řádku do chatu. Posílá se textový řetězec zprávy.
- *LineSend* – informace o postavené koleji. Posílá se id hrany a jméno hráče.
- *StoneSelect* – informace o postaveném základním kamenu. Posílá se id bodu a jméno hráče.
- *Ready* – po spuštění hry se čeká na ostatní spoluhráče. Hra se spustí, až jsou všichni hráči připraveni.
- *StartGame* – navazuje na zprávu Ready. Informuje o spuštění hry.

- *WinGame* – informace, jestli nějaký hráč propojil všech pět měst. Na serveru se provede aktualizace na nové kolo.
- *EndTurn* – značí ukončení tahu hráče. Posílá se jméno. Server aktualizuje hráče na tahu.
- *GetData* – hlavní zpráva, která se volá každou sekundu. Vrací aktuální data o hráčích a mapě.
- *GetDB* – tato zpráva požádá při spuštění hry o grafovou reprezentaci mapy z databáze.
- *GetMapDB* – podobně jako GetDB, ale vrací obrázek v bajtové podobě jako dlouhý textový řetězec
- *GetGamesDB* – tento požadavek získá data o všech hrách přihlášeného hráče z databáze. Posílá se pouze jméno hráče.
- *GetRecordsDB* – získá detailní data o jednotlivých tazích odehraných her z databáze. Posílá se id hry.
- *SelectPlayer* – vrací údaje o hráči z databáze. Posílá se jméno.
- *UpdatePlayer* – slouží pokud hráč aktualizuje své osobní údaje. Posílá se celý objet hráče ve formátu JSON.
- *Chyba na serveru* – poskytuje informaci o problémech na serveru.

8 Srovnání existujících řešení s touto práci

Hra TransAmerica, která je vytvořena v jazyce C# a kombinuje téměř všechny výhody obou zmíněných her (TransSib a Brettspielwel TransAmerica) v kapitole 3. Umožňuje hru po síti, ale také offline hru, tady hru bez nutnosti připojení k Internetu. Při vývoji byl kladen důraz na grafickou podobu (viz Obrázek 17), která věrohodně napodobuje deskovou hru. Hra také umožňuje načíst více map. Nevýhodou je podpora pouze pro platformu Windows a nutnost vytvoření databáze pro síťovou hru.

Obrázek 17: TransAmerica

Shrnutí kladů a záporů hry:

+ Hra má více herních režimů – možnost hrát hru online nebo offline.
+ Více map.
+ Umělý protivník.
+ Otevřený kód.
+ Moderní grafická stránka hry.
– Složitější správa serveru pro síťovou hru.
– Funguje pouze na operačním systému Windows.

Automatický protivník, který je implementován pro offline hru je deterministický. Pro stejné vstupy se chová vždy stejně. Nevýhodou je velmi snadná předvídatelnost, která města musí počítač propojit. Vždy staví k tomu městu, které je nejblíže k základnímu kameni. Tímto může hráč počkat, až postaví trať z jednoho konce mapy na druhý a pak se jen na trať napojí. Podobnou strategii ovšem může hráč uplatnit i proti živým protivníkům. Zde se nabízí prostor pro rozšíření aplikace. Dala by se implementovat částečná náhodnost výběru a tím by se vyhnulo přímočarosti počítače.

9 Závěr

Cílem této práce bylo navrhnout a implementovat deskovou hru TransAmerica. Aplikace je navržená podle zadání bakalářské práce a z vlastní iniciativy dále rozšířená o více režimů her pro offline i online. Protože jsou všechny algoritmy správně navržené, nebyl problém rozšířit aplikaci o více herních map.

Teoretická část bakalářské práce byla zaměřená na analýzu pravidel hry. Následovala analýza herní mapy a vhodného řešení pro její implementaci. Výsledkem bylo zjištění, že se jedná o grafovou síť, která se skládá z vrcholů a ohodnocených hran. Města a základní kameny byly vyhodnoceny jen jako speciální případy vrcholů. Zkušební implementace tuto analýzu následně vyhodnotily jako správnou.

V další části analýzy bylo nutno najít vhodné algoritmy pro práci s grafem. Přesněji algoritmy pro nelezení nejkratší cesty z bodu A do bodu B a nalezení hran na které může hráč stavět. Pro hledání nejkratší cesty bylo využito Dijkstrova algoritmu a pro dostupné hrany algoritmus průchodu grafem do hloubky. Jednou z nejzajímavějších částí této práce bylo navrhnout a implementovat automatického protivníka, který bude dostatečně nahrazovat živého hráče. Pro jeho logickou část bylo opět využito Dijkstrova algoritmu.

V části implementace již bylo nutno všechny tyto poznatky převést do kódu. Bylo využito objektového programovacího jazyka C#. Hra podle zadání měla umožňovat hru po síti, ale pro odzkoušení herní logiky byla nejprve zvolena implementace pro nesíťovou hru. Po zjištění, že je vše správně navrženo a vše funguje, se pokračovalo na implementaci herního serveru a vytvoření databáze. Pro komunikaci mezi serverem a klientem se využívá protokolu TCP/IP. Následně nastal problém při posílaní objektů, ale tento problém vyřešila serializace objektů do textového formátu JSON.

V poslední části práce bylo nutno vytvořit databázi, která se postupně rozšiřovala v průběhu implementace až do konečné podoby. Výsledkem je plně funkční hra v offline i online režimu pro jedno či více hráčů.

10 Reference

[1] Ocenění Japan Boardgame Prize [online]. [cit. 2016-02-04]. Dostupné z: https://boardgamegeek.com/wiki/page/Japan_Boardgame_Prize#

[2] Pravidla deskové hry TransAmerica [online] [cit. 2016-02-04]. Dostupné z: http://www.deskove-hry.eu/trans-america

[3] Herní server [online]. [cit. 2016-02-04]. Dostupné z: http://www.brettspielwelt.de/?nation=en

[4] Implementaci hry TransSib [online] [cit. 2016-02-04]. Dostupné z: https://sourceforge.net/projects/transsib/

[5] Dijkstra, Edsger W. "A note on two problems in connexion with graphs."Numerische mathematik 1.1 (1959): 269-271

[6] Tarjan, Robert. "Depth-first search and linear graph algorithms."SIAM journal on computing 1.2 (1972): 146-160

[7] Postel, Jon. "Transmission control protocol."(1981)

[8] Tsantalis, Nikolaos, et al. "Design pattern detection using similarity scoring."Software Engineering, IEEE Transactions on 32.11 (2006): 896-909.

[9] JSON fremwork pro .NET. [online] [cit. 2016-02-04]. Dostupné z: http://www.newtonsoft.com/json

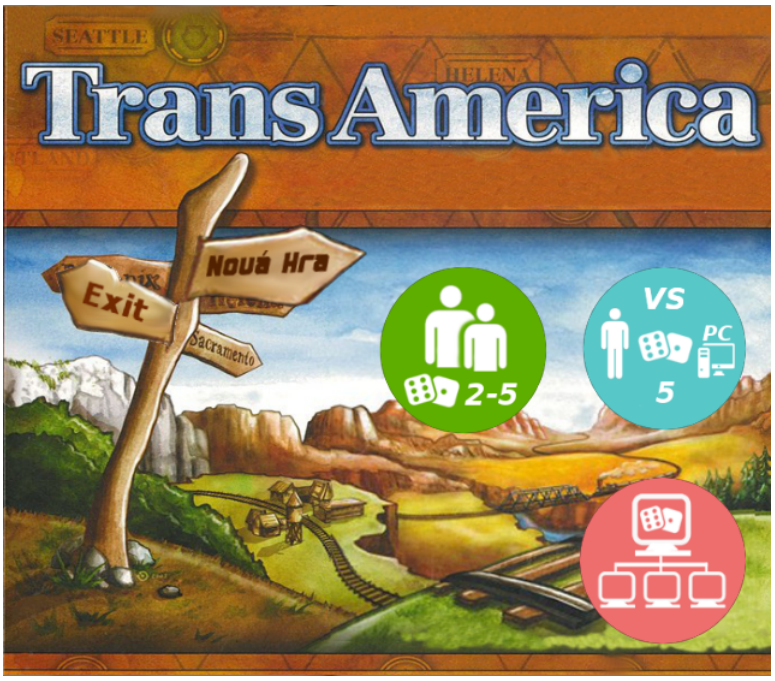
[10] Crockford, Douglas. "The application/json media type for javascript object notation (json)."(2006).

[11] Rumbaugh, James, Ivar Jacobson, and Grady Booch. Unified Modeling Language Reference Manual, The. Pearson Higher Education, 2004.

[12] Krasner, Glenn E., and Stephen T. Pope. "A description of the model-view-controller user interface paradigm in the smalltalk-80 system."Journal of object oriented programming 1.3 (1988): 26-49.

[13] XSS [online] [cit. 2016-02-04]. Dostupné z: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

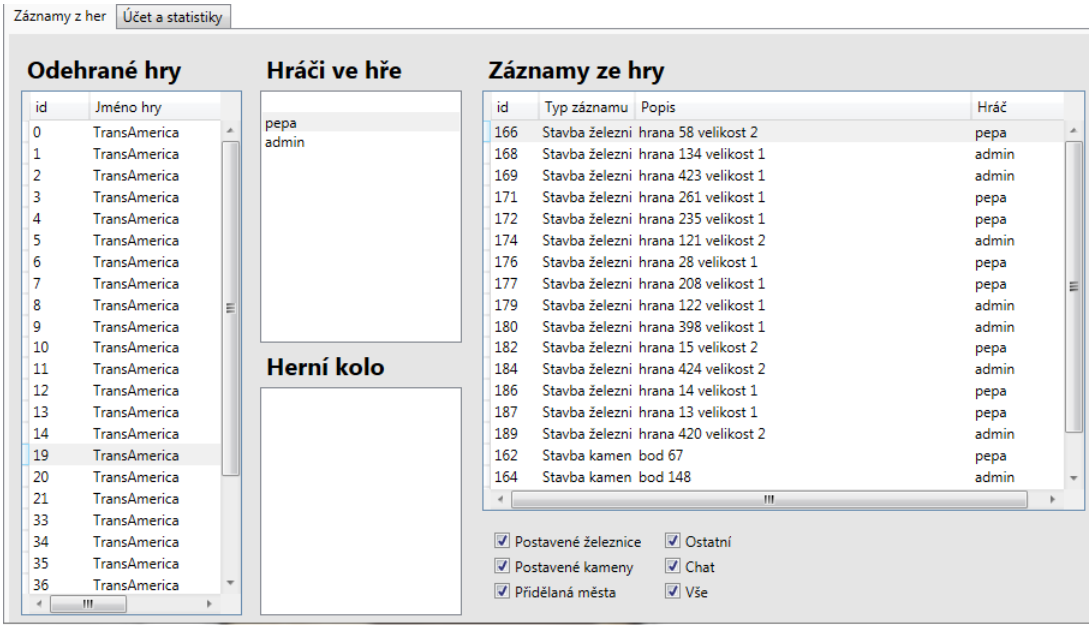
A Ukázky obrazovek GUI



Obrazek 18: Úvodní obrazovka – výběr typu hry

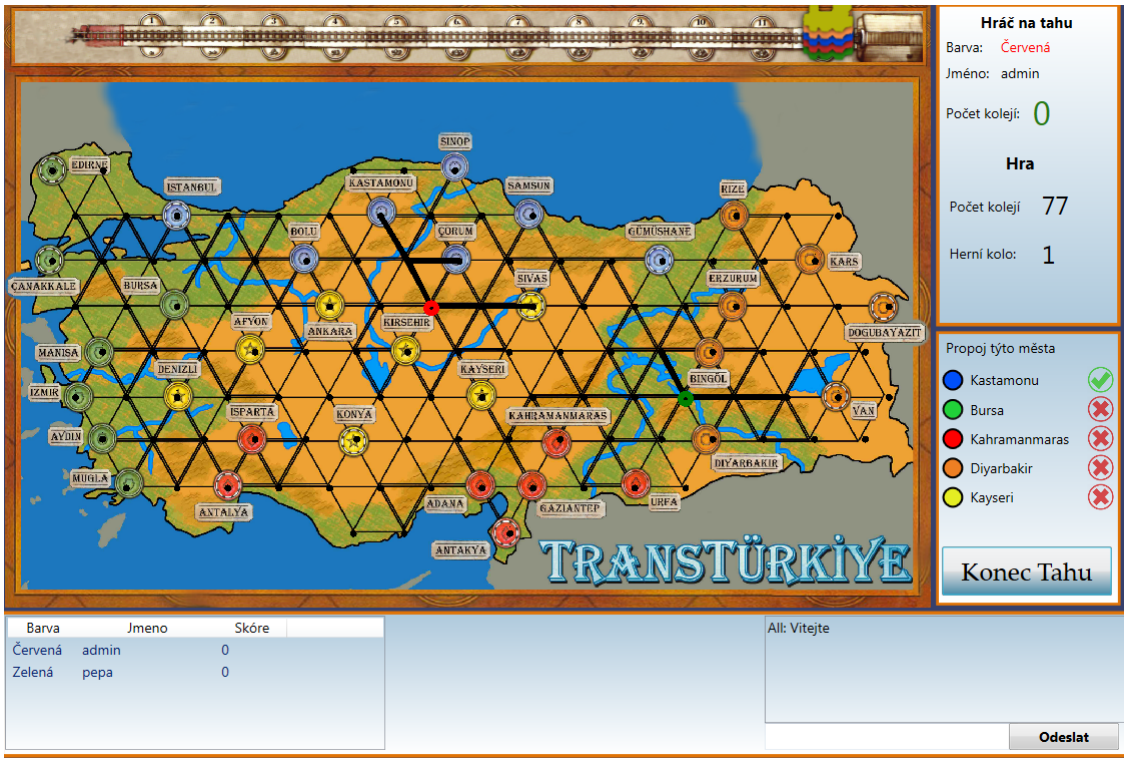
Obrazek 19: Výběr hry offline

Obrazek 20: Připojení online hry

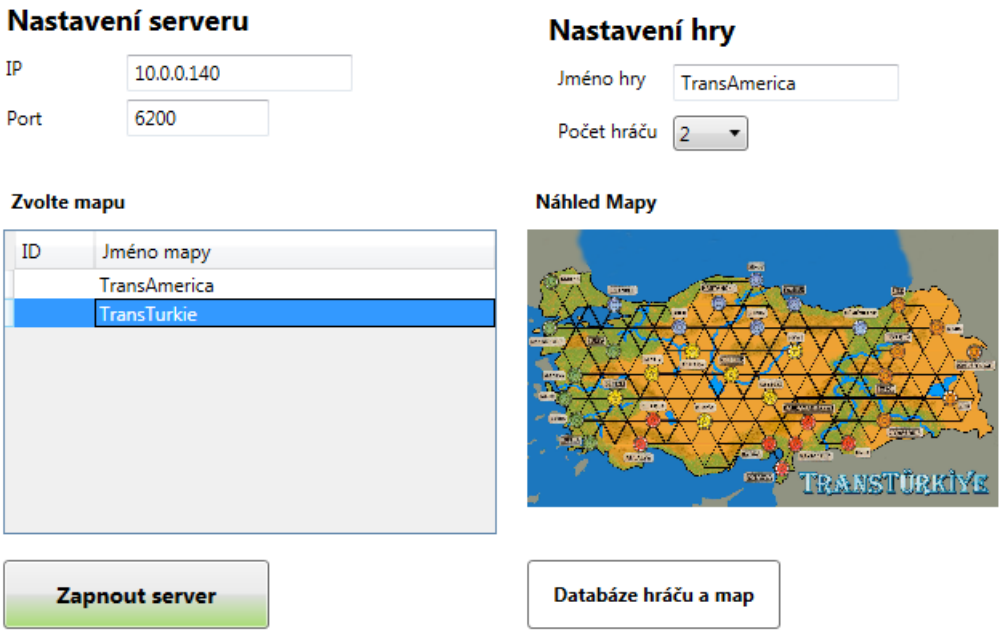


Obrazek 21: Záznamy z her

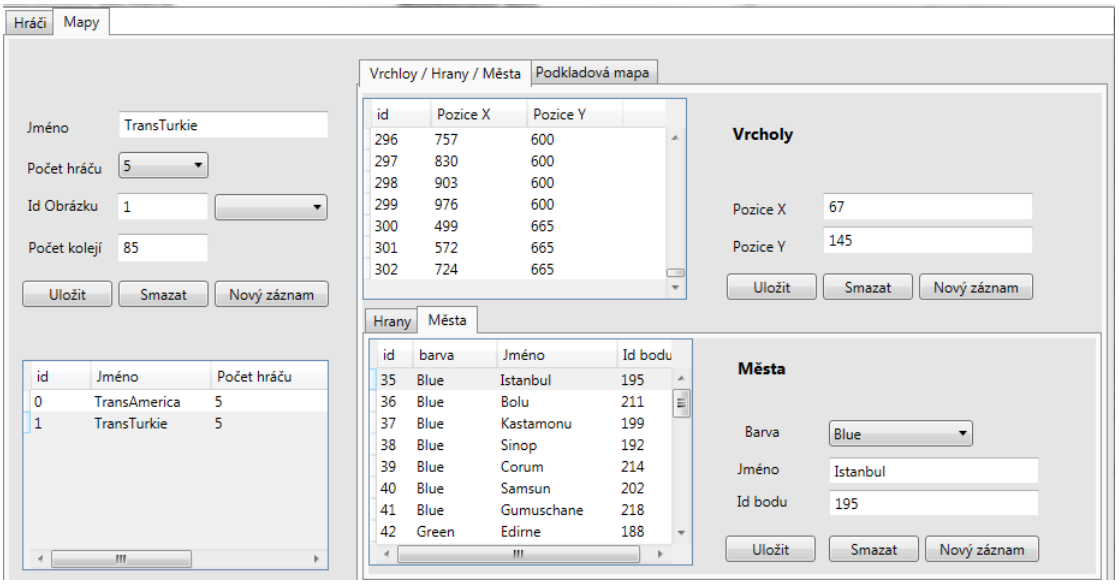
Obrazek 22: Ukázka herního okna offline



Obrazek 23: Ukázka herníáho okna online



Obrazek 24: Nastavení serveru



Obrazek 25: Správá databáze

B Příloha na CD/DVD

Obsah CD/DVD:

- *TransAmericaProjekt* – složka s projektem.
- *TransAmericaClient* – složka se spustitelným exe souborem Klienta.
- *TransAmericaServer* – složka se spustitelným exe souborem Serveru.
- *readme.txt* – pokyny pro správné spuštění aplikace.
- *Databáze* – složka se skripty pro vytvoření databáze.